

ARCHITECTURE DES ORDINATEURS
PARTIEL Octobre 2013 (2H)
Tous documents autorisés – Calculatrices autorisées.

PARTIE 1 : JEU D'INSTRUCTIONS MIPS32

Dans cette partie, on utilise le jeu d'instructions MIPS32.

On suppose que les registres R0 à R5 ont les contenus suivants, exprimés en hexadécimal :

R0	0000 0000
R1	8888 AAAA
R2	FEDC 3210
R3	0000 000C
R4	A800 0000
R5	C000 0000

Q 1) Donner les valeurs des registres modifiés après exécution des instructions suivantes.

- a) ADD R6, R1,R2
- b) ADDU R7, R4, R5
- c) ADD R8, R4, R5
- d) SUB R8, R5, R3
- e) SRA R9,R5,8
- f) SRL R10,R2,16

Q 2) Ecrire l'instruction ou la suite d'instructions qui place la constante 0x7FFFFFFF dans le registre R3

Q 3) On considère le programme suivant.

```
ADDI R1, R0, 0
BGTZ R2, suite // Branchement normal (non retardé)
ADDI R1, R1,-1
```

Suite : ADDI R1, R1, 1

Que contient le registre R1 après exécution du programme si R2 contient initialement

- a) FFFFFFFF
- b) 00000001

PARTIE 2 : JEU D'INSTRUCTIONS ARM

Dans cette partie, on utilise les instructions ARM décrites en annexe.

Q 4) Donner l'instruction pour multiplier le contenu du registre R0 par

- a) la constante 63
- b) la constante 17

Q5) Donner les instructions ARM pour effectuer

- Un décalage arithmétique à droite de R1 de 4 positions binaires
- Un décalage logique à droite de R1 de 2 positions binaires
- Un décalage logique à gauche de R1 de 4 positions binaires

Soit une zone mémoire à partir de l'adresse C000 0000

Adresse (hexadécimal)	Contenu mot 32 bits (hexadécimal)
C000 0000	0x 12 34 56 78
C0000 0004	0x FE DC BA 98
C000 0008	0x 00 11 22 33
C000 000C	0x A1 B2 C3 D4
C000 0010	0x F9 E8 D7 C6

Q 6) Donner le contenu des registres ou des cases mémoire modifiées (MEM8, MEM16 ou MEM32) après exécution des instructions suivantes. On suppose l'ordre « little endian ». Le registre R1 contient 0x33335555, le registre R3 contient 0x00000008 et le registre R5 contient 0xC000 0000.

- a) LDRSB R11, [R5, #4]
- b) LDRH R12, [R5, R3]
- c) LDR R13, [R5, #12]
- d) LDR R14, [R5], #4
- e) LDR R6, [R5, #8] !
- f) STR R1, [R5], #8
- g) STRB R3, [R5, -R3]
- h) LDR R7, [R5, #6] !

NB : l'ordre « little endian » range le mot 0x0A0B0C0D dans l'ordre suivant dans les 4 premiers octets de la mémoire :

Octet d'adresse 3 : 0A - Octet d'adresse 2 : 0B - Octet d'adresse 1 : 0C - Octet d'adresse 0 : 0D

L'octet d'adresse 0 est donc 0D. Le mot de 16 bits d'adresse 0 est donc 0C0D. Le mot de 32 bits d'adresse 0 est donc 0A0B0C0D

Q 7) Soit le programme ARM qui effectue un traitement sur deux tableaux d'entiers X et Y. Donner le programme C correspondant au programme assembleur ARM. Quel est le résultat de l'exécution du programme.

```

LDR r1, =X
LDR r2, =Y
MOV r3, #10
LOOP: LDR r4, [r1], #4
      LDR r5, [r2], #4
      LDR r6, [r1], #4
      LDR r7, [r2], #4
      ADD r0, r5, r4
      STR r0, [r2, #-8]
      ADD r0, r7, r6
      STR r0, [r2, #-4]
      SUBS r3, r3, #2
      BGT LOOP
      SWI 0x11 @ Stop program execution

```

```
.data
X: .word 1, -3, 5, -7, 9, -11, 13, -15, 17, -19
Y: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

Q 8) Ecrire un programme ARM qui compte le nombre d'entiers strictement positifs dans un tableau de 10 entiers X. Le résultat sera retourné dans le registre R0.

Q9) Ecrire un programme ARM qui compte le nombre de bits à 1 dans le registre R1 et met le résultat dans R0.

Annexe : Jeu d'instructions ARM

On rappelle que le processeur ARM a 15 registres de 32 bits. Les immédiats sont signés. R15 est le compteur de programme.

Instruction	Assembleur	Effet
MOV	MOV Ri,Rj MOV Ri,Rj Décalage #N	Ri ← Rj Ri ← Rj décalé de N positions
ADD	ADD Ri, Rj, Rk ADD Ri, Rj, #N ADD Ri, Rj, Rk Décalage #N	Ri ← Rj+Rk Ri ← Rj+N Ri ← Rj+(Rk décalé de N positions)
SUB	SUB Ri, Rj, Rk SUB Ri, Rj, #N SUB Ri, Rj, Rk Décalage #N	Ri ← Rj - Rk Ri ← Rj - N Ri ← Rj - (Rk décalé de N positions)
RSB	RSB Ri, Rj, Rk RSB Ri, Rj, #N RSB Ri, Rj, Rk Décalage #N	Ri ← Rk - Rj Ri ← N - Rj Ri ← (Rk décalé de N positions) -Rj
AND (et logique)	AND Ri, Rj, Rk AND Ri, Rj, #N AND Ri, Rj, Rk Décalage #N	Ri ← Rj and Rk Ri ← Rj and N Ri ← Rj and (Rk décalé de N positions)
SUBS	SUBS Ri, Rj, Rk SUBS Ri, Rj, #N SUBS Ri, Rj, Rk Décalage #N	Ri ← Rj - Rk et Rcc positionné Ri ← Rj - N et Rcc positionné Ri ← Rj - (Rk décalé de N positions) et Rcc positionné
ORR (ou logique)	ORR Ri, Rj, Rk ORR Ri, Rj, #N ORR Ri, Rj, Rk Décalage #N	Ri ← Rj or Rk Ri ← Rj or N Ri ← Rj or (Rk décalé de N positions)
EOR (ou exclusif)	EOR Ri, Rj, Rk EOR Ri, Rj, #N EOR Ri, Rj, Rk Décalage #N	Ri ← Rj xor Rk Ri ← Rj xor N Ri ← Rj xor (Rk décalé de N positions)
CMP	CMP Ri,Rj CMP Ri, #N	Compare Ri et Rj (ou Ri et #N) et positionne le registre code condition
BGT	BGT adresse cible	Branchement si le résultat de la comparaison (CMP) était >0 ou si le résultat de SUBS >0

Table 1 : Opérations arithmétiques et logiques utilisées

Les instructions mémoire utilisées sont données dans la table 2. L'adresse mémoire est donnée par le mode d'adressage indiqué dans la table 3. Mem32 signifie un accès mémoire à un mot de 32 bits. Mem16 signifie un accès mémoire à un demi-mot (16 bits). Mem8 signifie un accès octet. Dans le cas d'un accès Mem16 et Mem8, il est précisé si le registre de 32 bits est complété à gauche par des 0 (extension zéro) ou par le signe du demi-mot ou de l'octet lu (extension signe).

Instruction	Effet	Commentaire
LDR	$Rn \leftarrow \text{Mem32 (Adresse)}$	Chargement mot
LDRH	$Rn \leftarrow \text{extension zéro, Mem16 (Adresse)}$	Chargement demi mot non signé
LDRSH	$Rn \leftarrow \text{extension signe, Mem16 (Adresse)}$	Chargement demi mot signé
LDRB	$Rn \leftarrow \text{extension zéro, Mem8 (Adresse)}$	Chargement octet non signé
LDRSB	$Rn \leftarrow \text{extension signe, Mem8 (Adresse)}$	Chargement octet signé
STR	$\text{Mem32 (Adresse)} \leftarrow Rn$	Rangement mot
STRH	$\text{Mem16 (Adresse)} \leftarrow Rn[15:0]$	Rangement demi mot
STRB	$\text{Mem8 (Adresse)} \leftarrow Rn[7:0]$	Rangement octet

Table 2 : Instructions mémoire

Mode	Assembleur	Action
Déplacement 12 bits, Pré-indexé	$[Rn, \#d\text{eplacement}]$	Adresse = $Rn + d\text{eplacement}$
Déplacement 12 bits, Pré-indexé avec mise à jour	$[Rn, \#d\text{eplacement}] !$	Adresse = $Rn + d\text{eplacement}$ $Rn \leftarrow \text{Adresse}$
Déplacement 12 bits, Post-indexé	$[Rn], \#d\text{eplacement}$	Adresse = Rn $Rn \leftarrow Rn + d\text{eplacement}$
Déplacement dans Rm Préindexé	$[Rn, \pm Rm, d\text{ecalage}]$	Adresse = $Rn + d\text{ecalage} (Rm)$
Déplacement dans Rm Préindexé avec mise à jour	$[Rn, \pm Rm, d\text{ecalage}] !$	Adresse = $Rn + d\text{ecalage} (Rm)$ $Rn \leftarrow \text{Adresse}$
Déplacement dans Rm Postindexé	$[Rn], \pm Rm, d\text{ecalage}$	Adresse = Rn $Rn \leftarrow Rn + d\text{ecalage} (Rm)$

Table 3 : Modes d'adressage.